Statistical learning (MT7038) - Project 1

Sergio Arnaud

16 december, 2019

K-means

Task 1

This task allows you to experience the performance of k-mean clustering and its possible limitations when applying to imbalance data and non-spherical symmetric data. The data for this task are from the text files "Imbalance_Data.txt" and "Star_Data.txt". The first and second columns of the data files are the x1 and x2 coordinates of the 2D feature vector, respectively.

a) *For the "Star_Data.txt" data, write a simple code (you can use the kmeans() function in R) to cluster the data with number of clusters Nc = 2, 3, and 4. Plot the clustering results on the x_1 and x_2 plane for $N_c = 2, 3,$ and 4.*

We begin by reading the data from `Star_Data.txt`

```
star_data = read.table('Star_Data.txt', col.names = c('x1','x2'))
summary(star_data)
```

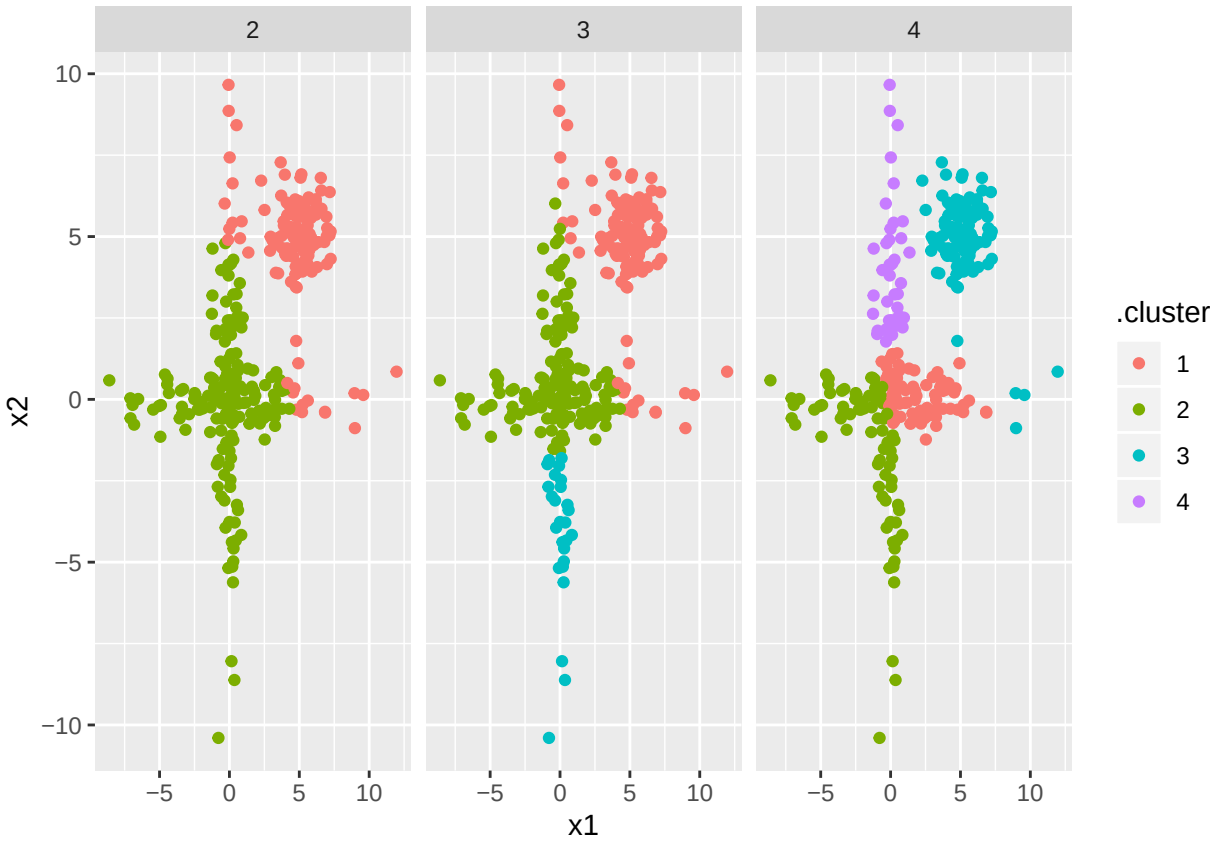
```
##        x1                x2          
##  Min.   :-8.5962   Min.   :-10.3979  
##  1st Qu.:-0.1007   1st Qu.: -0.2115  
##  Median : 0.9134   Median :  1.0325  
##  Mean   : 1.8460   Mean   :  1.9163  
##  3rd Qu.: 4.7281   3rd Qu.:  4.7671  
##  Max.   :11.9589   Max.   :  9.6605
```

The following function, for every number of clusters from 2 to 4 it computes the kmean model and predict the cluster categories for each data point of the star dataset.

```
assignments <- tibble(k = 2:4) %>%
  mutate(
    model = map(k, ~kmeans(star_data, .x)),
    augmented = map(model, augment, star_data)
  ) %>%
  unnest(augmented)
```

Finally, we plot the clustering results for the number of clusters Nc = 2, 3, and 4.

```
ggplot(assignments, aes(x1, x2)) +
  geom_point(aes(color = .cluster)) +
  facet_wrap(~ k)
```



---

b) *Discuss the performance of the k-mean clustering from the plots in a). If the clustering results do not look good, suggest a reasonable way for improvement and justify your answer.*

I don't think k-mean is the optimal clutering algorithm for this case. Indeed, not only the results don't look good but this is a case of non-spherical data. K-means does not perform well when the groups are grossly non-spherical since the algorithm is trying to find centers and spheres around them such that it minimizes the sum of squares. The algorithm it's still minimizing the within-cluster sum of squares, but not in the best way for this data.

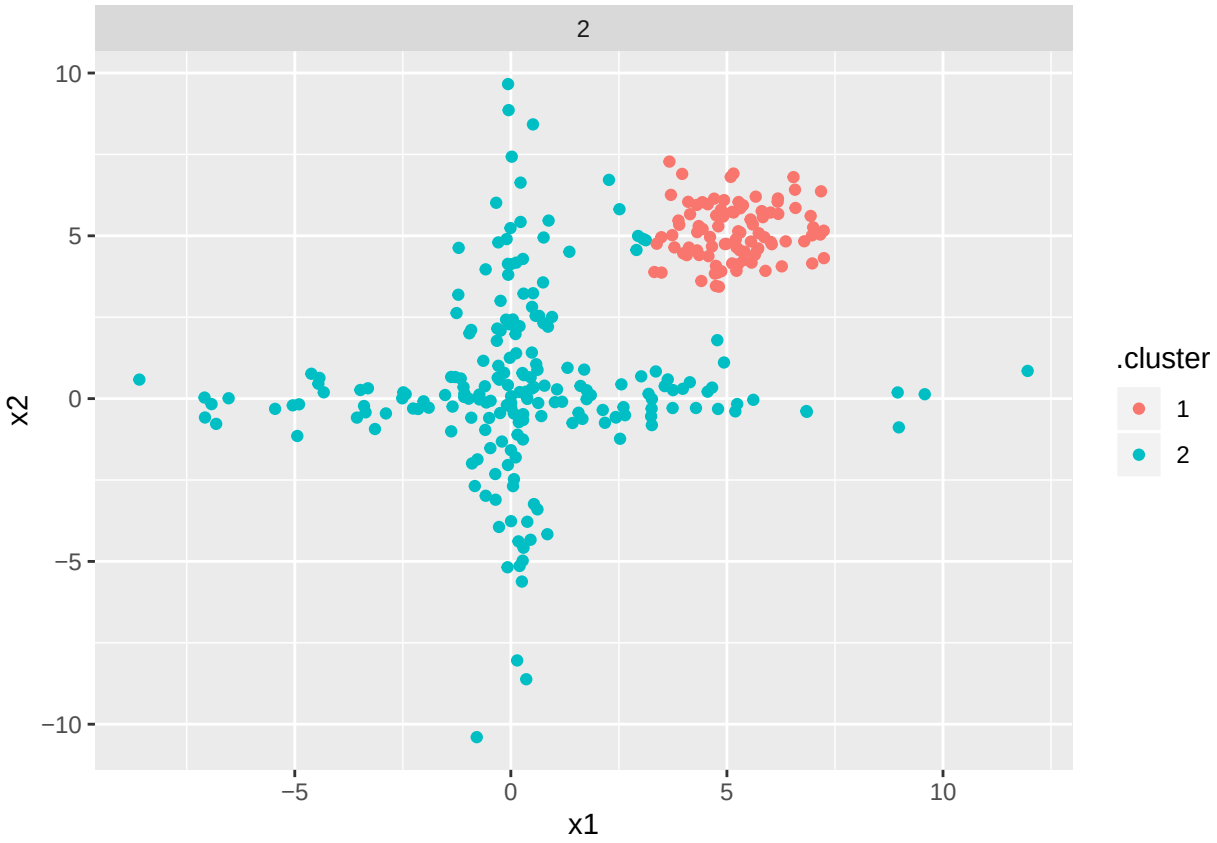
Said that, we can still apply transformations or introduce new variables if we want the algorithm to detect something in particular. In this case, I would say that the data has 2 clusters, the points that lie in the x and y axis forming the + sign and the points forming an spherical cluster with it's center roughly at $(5, 5)$. With that in mind we could introduce a new variable $z = min(x^2, y^2)$ so that we can try to quantify the distance for any point to it's closest axis.

```
star_data <- star_data %>% mutate(z = map2(x1,x2,~min(.x^2,.y^2) ))
```

We can now, apply k-means again and look at the results

```
assignments <- tibble(k = 2:2) %>%
  mutate(
    model = map(k, ~kmeans(star_data, .x)),
    augmented = map(model, augment, star_data)
  ) %>%
  unnest(augmented)
ggplot(assignments, aes(x1, x2)) +
```

```
  geom_point(aes(color = .cluster)) +
  facet_wrap(~ k)
```



It looks that in this case, k-means splitted fairly good those clusters. Transforming and augmenting operations might solve the problem of non-spherical data for k-means.

---

c) *Repeat part a) and b) above using the data file "Imbalance_Data.txt" . ****

We begin by loading the data from the text file `Imbalance_Data.txt`

```
imbalanced_data = read.table('Imbalance_Data.txt', col.names = c('x1','x2'))
summary(imbalanced_data)
```

```
##        x1                x2          
##  Min.   :-4.932   Min.   :-2.95004  
##  1st Qu.:-2.723   1st Qu.:-0.62576  
##  Median :-1.976   Median :-0.04664  
##  Mean   :-1.852   Mean   :-0.02846  
##  3rd Qu.:-1.262   3rd Qu.: 0.52938  
##  Max.   : 2.620   Max.   : 3.09797
```

For every number of clusters from 2 to 4, compute the kmean model and predict the cluster categories for each data point of the imbalanced dataset.

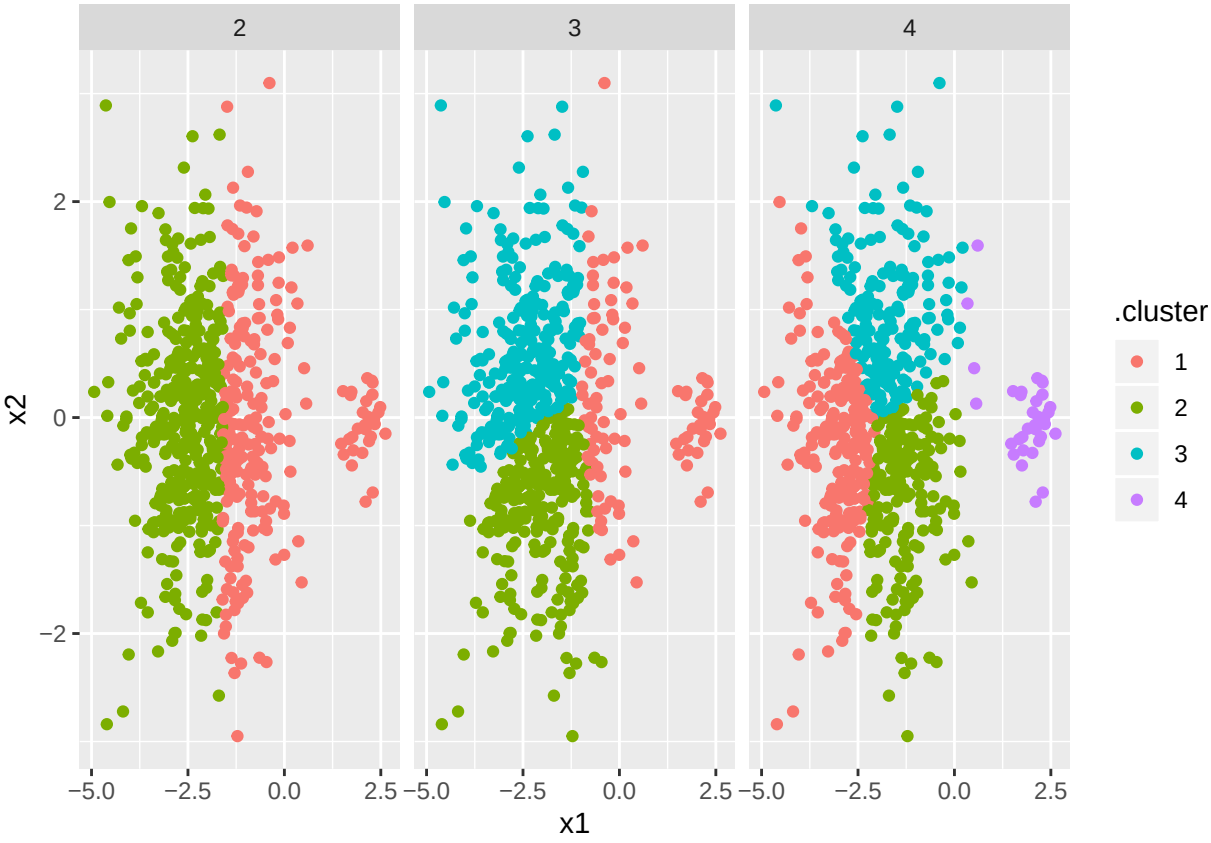
```
assignments <- tibble(k = 2:4) %>%
  mutate(
    clustering = map(k, ~kmeans(imbalanced_data, .x)),
    augmented = map(clustering, augment, imbalanced_data)
```

```
  ) %>%
  unnest(augmented)
assignments
```

```
## # A tibble: 1,890 x 5
##        k clustering    x1      x2 .cluster
##    <int> <list>     <dbl>   <dbl> <fct>
##  1     2 <kmeans>   -1.74  0.453  2
##  2     2 <kmeans>   -2.05  2.06   2
##  3     2 <kmeans>   -2.11  0.214  2
##  4     2 <kmeans>   -1.59 -0.942  1
##  5     2 <kmeans>   -3.17  0.0123 2
##  6     2 <kmeans>   -2.26  0.604  2
##  7     2 <kmeans>   -2.19 -0.787  2
##  8     2 <kmeans>   -1.20 -1.02   1
##  9     2 <kmeans>   -2.41 -0.278  2
## 10     2 <kmeans>   -1.32 -0.660  1
## # ... with 1,880 more rows
```

Finally, we plot the clustering results for the number of clusters Nc = 2, 3, and 4.

```
ggplot(assignments, aes(x1, x2)) +
  geom_point(aes(color = .cluster)) +
  facet_wrap(~ k)
```



Once again, the resulting clusters don't appear to resable the data. In this case the problem is that we are working with unbalanced clusters of data.

In particular, I would say that we have two clusters, the first one has the point $(2.5, 0)$ as center and much

more variance and observations. The second one has the point $(-2.5, 0)$ as center and much less variance and observations. Normalizing the coordinates could be a succesfull transformation.

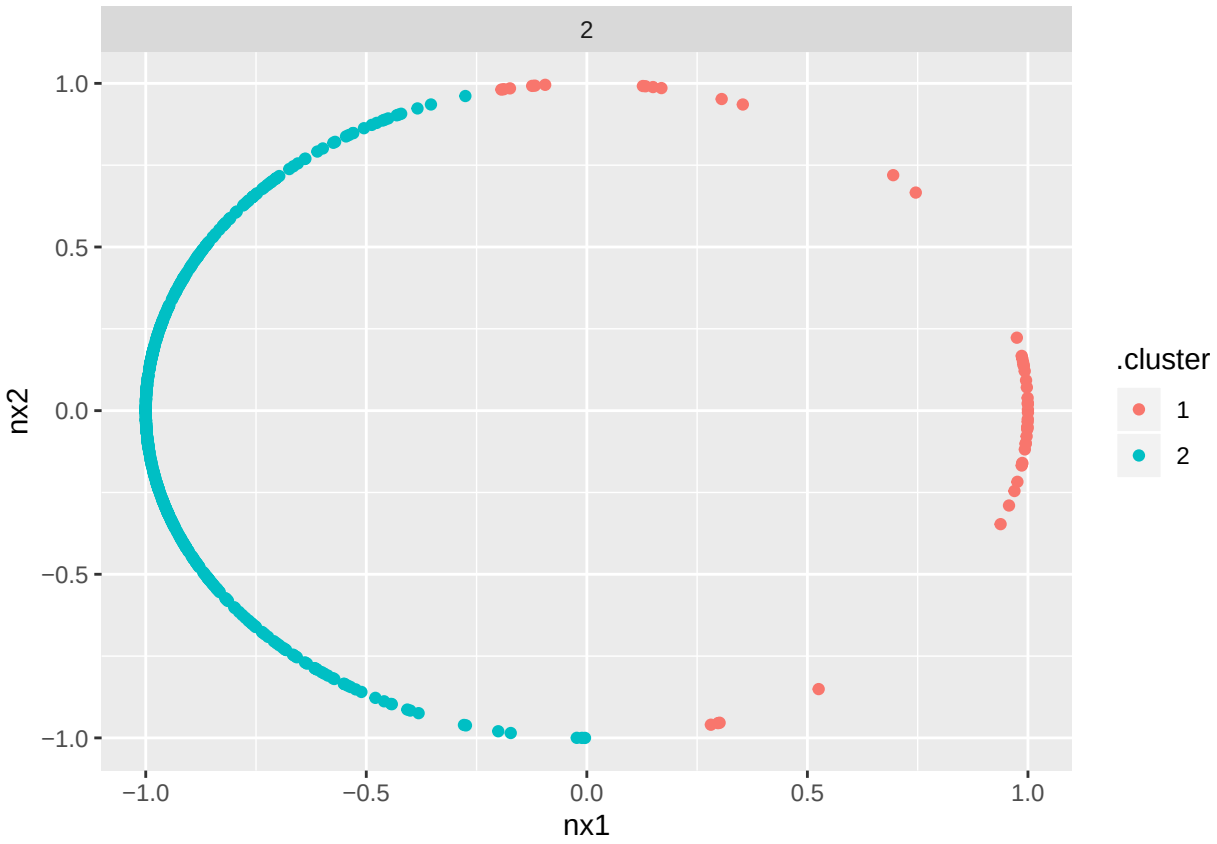
```
imbalanced_data2 <- imbalanced_data %>% mutate(
                                          nx1 = x1/sqrt(x1^2+x2^2),
                                          nx2 = x2/sqrt(x1^2+x2^2)
                                            )
```

Now we compute the clusters and plot the results

```
assignments <- tibble(k = 2) %>%
  mutate(
    clustering = map(k, ~kmeans(imbalanced_data2 %>% select(nx1, nx2), .x)),
    augmented = map(clustering, augment, imbalanced_data2)
  ) %>%
  unnest(augmented)
assignments
```

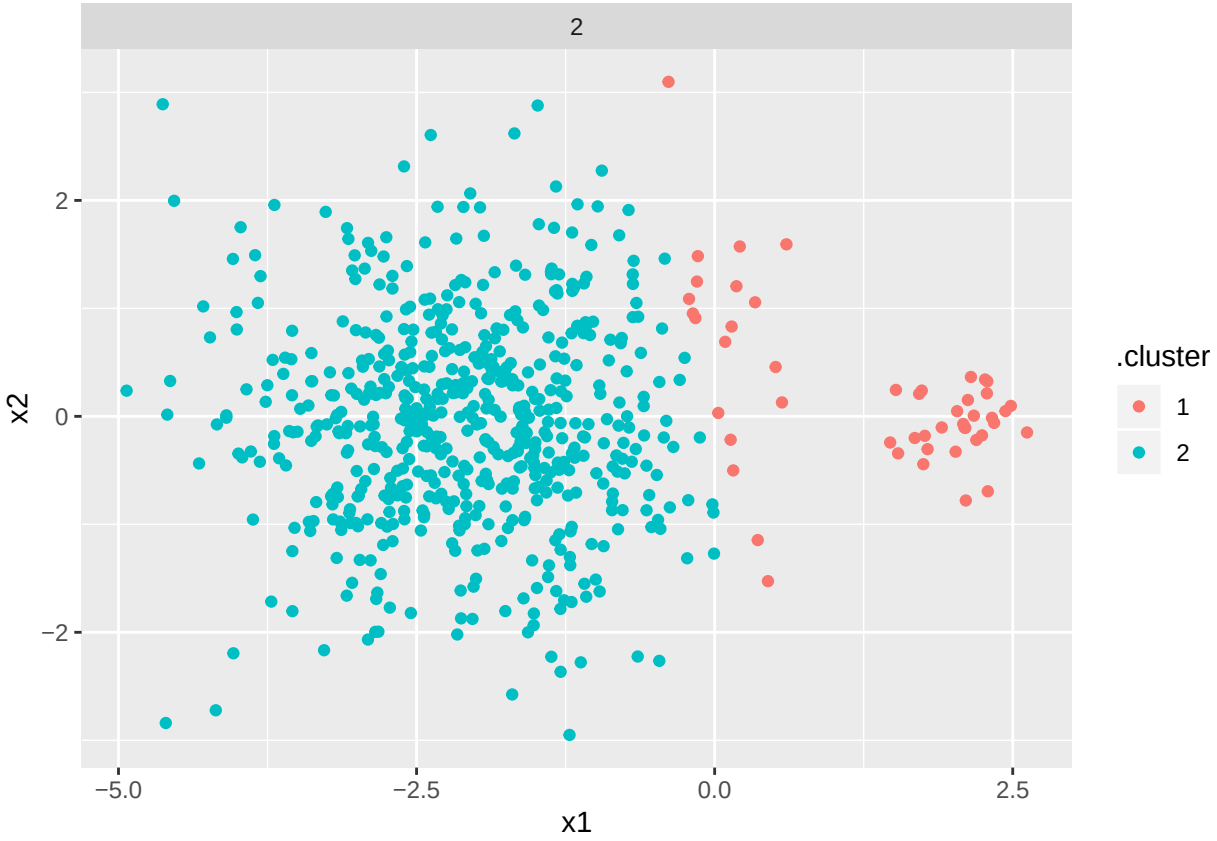
```
## # A tibble: 630 x 7
##        k clustering    x1      x2    nx1      nx2 .cluster
##    <dbl> <list>     <dbl>   <dbl>  <dbl>    <dbl> <fct>
##  1     2 <kmeans>   -1.74  0.453  -0.968  0.251   2
##  2     2 <kmeans>   -2.05  2.06   -0.705  0.710   2
##  3     2 <kmeans>   -2.11  0.214  -0.995  0.101   2
##  4     2 <kmeans>   -1.59 -0.942  -0.861 -0.509   2
##  5     2 <kmeans>   -3.17  0.0123 -1.000  0.00387 2
##  6     2 <kmeans>   -2.26  0.604  -0.966  0.258   2
##  7     2 <kmeans>   -2.19 -0.787  -0.941 -0.339   2
##  8     2 <kmeans>   -1.20 -1.02   -0.761 -0.649   2
##  9     2 <kmeans>   -2.41 -0.278  -0.993 -0.114   2
## 10     2 <kmeans>   -1.32 -0.660  -0.894 -0.448   2
## # ... with 620 more rows
```

```
ggplot(assignments, aes(nx1, nx2)) +
  geom_point(aes(color = .cluster)) +
  facet_wrap(~ k)
```



The former plot was made with the transformed data, before normalization it looks in the following way.

```
ggplot(assignments, aes(x1, x2)) +
  geom_point(aes(color = .cluster)) +
  facet_wrap(~ k)
```



That looks to be a much better result than the one obtained before normalization.

K-means

Task 2 *In this task, you will implement PCA as an eigen-problem and understand its difference from linear regression, besides one is unsupervised and one is supervised. This task uses the data from the file "PCA_Data.txt". The first and second columns of the data file are the x1 and x2 coordinates of the 2D feature vector, respectively.*

a) *Write a simple code to compute the 2 PCs from the data points. The code should perform a diagonalization of the corresponding covariance matrix to find the eigenvectors. The built-in R functions* `prcomp()` *and* `princomp()` *should not be used in this exercise. Plot the 2 PCs together with the data points. Note: Make sure to center the data points correctly such that the PCs should go through the mean location of the data points.*

Loading the data from the text file `PCA_data.txt`

```
pca_data = read.table('PCA_data.txt', col.names = c('x1','x2'))
summary(pca_data)
```

```
##        x1                x2
##  Min.   :0.005014   Min.   : 1.373
##  1st Qu.:2.496765   1st Qu.: 8.732
##  Median :3.836865   Median :11.554
##  Mean   :4.390614   Mean   :13.320
##  3rd Qu.:6.965760   3rd Qu.:20.634
```

```
##  Max.   :9.285430   Max.   :25.513
```

Centering the data points

```
col_mean <- map(pca_data, mean)

pca_data <- pca_data %>%
    map2_df(col_mean, ~.x - .y)
```

Computing the SVD for the covariance matrix of our dataset

$$Cov(X) = UDV^*$$

```
s <- svd(cov(pca_data))

U <- s$u
D <- diag(s$d)
V <- s$v
```

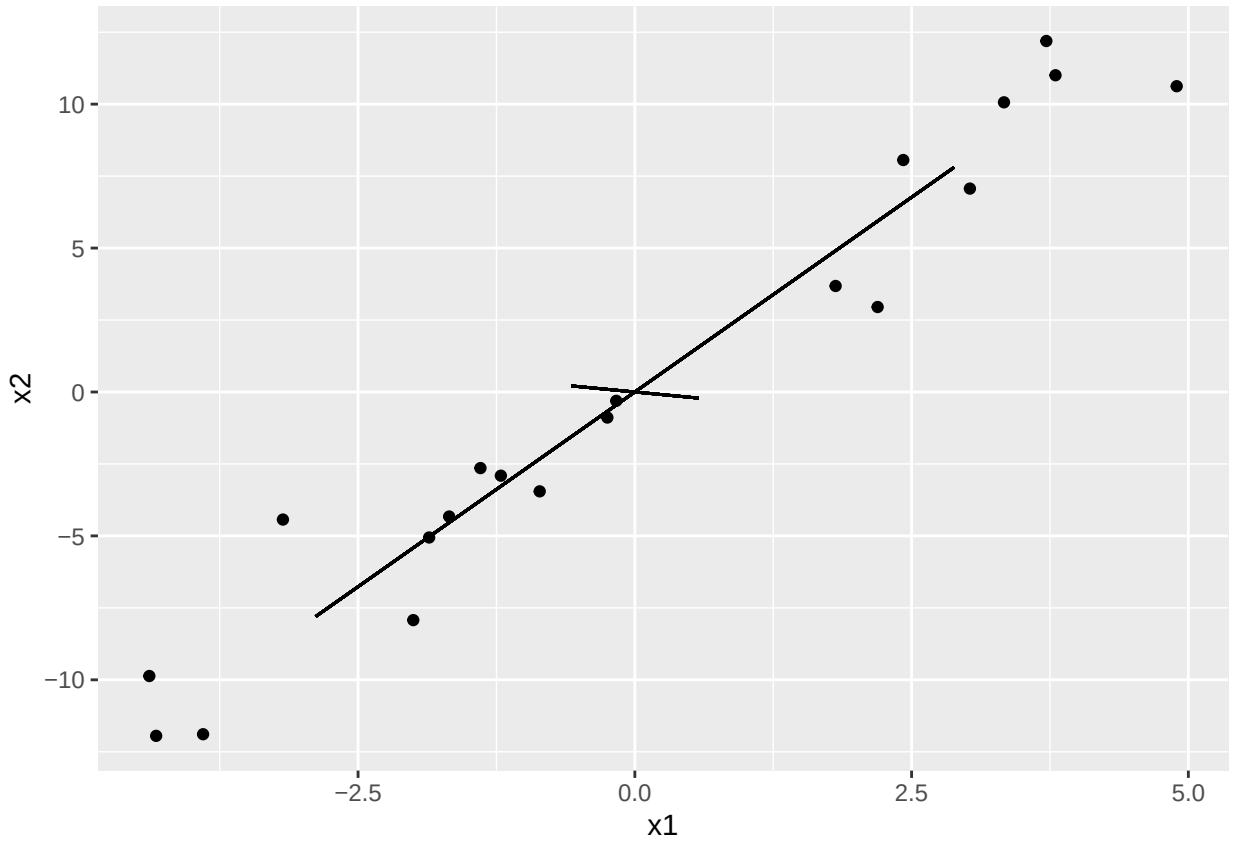
And computing the principal components out of the U and D

```
pcs <- U %*% sqrt(D)
pcs
```

```
##           [,1]       [,2]
## [1,] -2.883311 -0.5742342
## [2,] -7.802272  0.2122069
```

Plotting the two principal components together with the data points

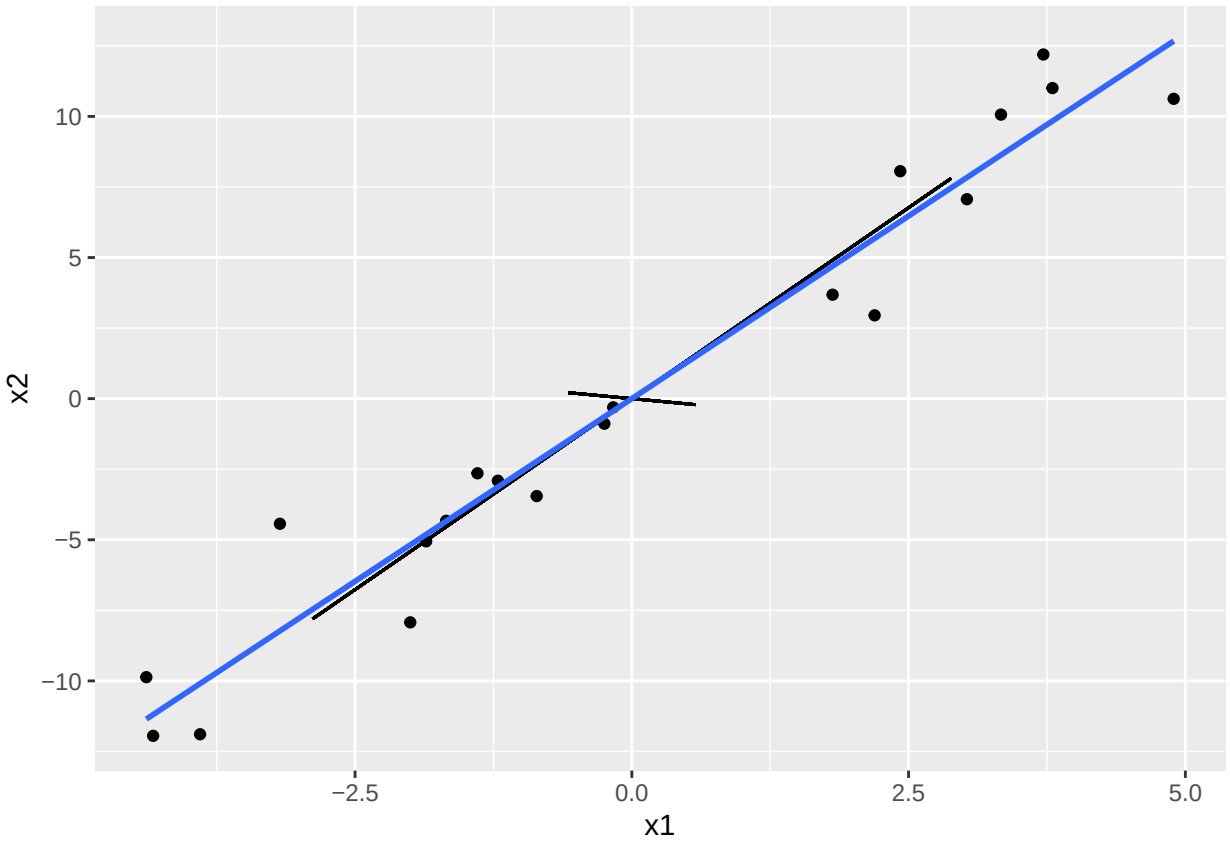
```
ggplot(pca_data, aes(x1, x2)) +
  geom_point() +
  geom_segment(aes(x = -pcs[1,1], y = -pcs[2,1], xend = pcs[1,1], yend = pcs[2,1])) +
  geom_segment(aes(x = -pcs[1,2], y = -pcs[2,2], xend = pcs[1,2], yend = pcs[2,2]))
```



---

b) *Now treat the x2 and x1 coordinates of the data as the response and predictor variables, respectively, and perform a linear regression for the data using the least square fitting (i.e., a supervised learning). Plot the linear regression line together with the first PC and the data points. Is the linear regression line the same as the first PC?*

```
ggplot(pca_data, aes(x1, x2)) +
  geom_point() +
  geom_segment(aes(x = -pcs[1,1], y = -pcs[2,1], xend = pcs[1,1], yend = pcs[2,1])) +
  geom_segment(aes(x = -pcs[1,2], y = -pcs[2,2], xend = pcs[1,2], yend = pcs[2,2])) +
  geom_smooth(method='lm', se = FALSE) #Computing a linear regression and plotting it
```



As we can see, the linear regression line is very similar to the one of the first principal component, nevertheless, they aren't the same.

SVM

Task 3 *This task allows you to experience the usage of svm() function in R and explore a little bit how to generalize to nonlinear decision boundary. This task uses the "mixture simulation" data in the course book. To download the data, go to the webpage of the course book (https://web.stanford.edu/~hastie/ElemStatLearn/), and then click "Data" on the left panel to find the data and follow the instruction to use them.*

a) *As a warm up exercise, perform the support vector machine for linear classification for the data using the svm() function. To use the svm() function, you need to first install the R-package e1071 (run install.package("e1071") to install). Re- produce the two figures in Fig. 12.2 in the course book with $C = 10000$ and $C = 0.01$. Hint: use the C-classification and the linear kernel in the svm() function.*

First, we load the data

```
load(file = "ESL.mixture.rda")
glimpse(ESL.mixture)
```

```
## List of 8
##  $ x       : num [1:200, 1:2] 2.5261 0.367 0.7682 0.6934 -0.0198 ...
##  $ y       : num [1:200] 0 0 0 0 0 0 0 0 0 0 ...
##  $ xnew    : 'matrix' num [1:6831, 1:2] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -1.9 -1.8 -1.7 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:6831] "1" "2" "3" "4" ...
```

```
##   .. ..$ : chr [1:2] "x1" "x2"
##  $ prob    : num [1:6831] 3.55e-05 3.05e-05 2.63e-05 2.27e-05 1.96e-05 ...
##   ..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
##  $ marginal: num [1:6831] 6.65e-15 2.31e-14 7.62e-14 2.39e-13 7.15e-13 ...
##   ..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
##  $ px1     : num [1:69] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -1.9 -1.8 -1.7 ...
##  $ px2     : num [1:99] -2 -1.95 -1.9 -1.85 -1.8 -1.75 -1.7 -1.65 -1.6 -1.55 ...
##  $ means   : num [1:20, 1:2] -0.2534 0.2667 2.0965 -0.0613 2.7035 ...
```

We will create a new dataframe with the predictors and categories called `mixture_train` and another one, named `mixture_test` that contains 6831 lattice points in the predictor space.

```
mixture_train <- data.frame(
                            x1 = ESL.mixture$x[,1],
                            x2 = ESL.mixture$x[,2],
                            y = as.factor(ESL.mixture$y)
                           )

mixture_test <- data.frame( x1 = ESL.mixture$xnew[,1],
                      x2 = ESL.mixture$xnew[,2])
```

Let's take a quick look of our data

```
ggplot(mixture_train, aes(x1,x2, colour = y)) +
    geom_point()
```

Let's extract some more information from the `ESL.mixture`so that we can calculate the bayes and test errors

```
marginal <- ESL.mixture$marginal
prob <- ESL.mixture$prob
```

For this data, the bayes error is calculated as follows

```
bayes.error<-sum(marginal*(prob*I(prob<0.5)+(1-prob)*I(prob>=.5)))
bayes.error
```

```
## [1] 0.2101192
```

The following function work as follows: given a model, it computes the predictions (as a probability) of such model for the 6831 lattice points in the predictor space. The, it computes and return the test error for such model.

```
get_test_error <- function(model){
  pred <- predict(model, mixture_test, probability=TRUE)
  pred <- logit(attr(pred, 'probabilities')[,2])
  test.error<-sum(marginal*(prob*I(pred < 0)+(1-prob)*I(pred > 0)))
  round(test.error,3)
}
```

Now, we build a dataframe `results` that, for each cost, trains the svm model and computes the train and test errors.

```
results <- tibble(cost = c(.01, 10000)) %>%
  mutate(
    model = map(cost, ~svm(y ~ ., mixture_train, type = 'C-classification',
                                kernel = 'linear', cost=.x, probability=TRUE)),

    train_predicted = map(model, ~predict(.x, mixture_train %>% select(x1,x2))),
    train_actual = map(model, ~identity(mixture_train$y)),
    train_correct = map2_dbl(train_predicted, train_actual, ~sum(.x == .y)),
    train_error = (nrow(mixture_train)-train_correct)/nrow(mixture_train),

    test_predicted = map(model, ~predict(.x, mixture_test %>% select(x1,x2))),

    test_actual = map(model, ~identity(mixture_test$y)),
    test_error = map(model, ~get_test_error(.x))
  )
glimpse(results)
```

```
## Observations: 2
## Variables: 9
## $ cost            <dbl> 1e-02, 1e+04
## $ model           <list> [<svm(formula = y ~ ., data = mixture_train, type ...
## $ train_predicted <list> [<0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ train_actual    <list> [<0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ train_correct   <dbl> 147, 146
## $ train_error     <dbl> 0.265, 0.270
## $ test_predicted  <list> [<0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ test_actual     <list> [NULL, NULL]
## $ test_error      <list> [0.294, 0.291]
```

Finally, we will build a grid in the feature space

```
grid <- expand.grid(x1 = seq(from = min(mixture_train$x1), by = .025, to = max(mixture_train$x1) ),
                    x2 = seq(from = min(mixture_train$x2), by = .025, to = max(mixture_train$x2) ))
```

For each point (x, y) in the grid, we'll calculate the probability that such point belong to the first class and

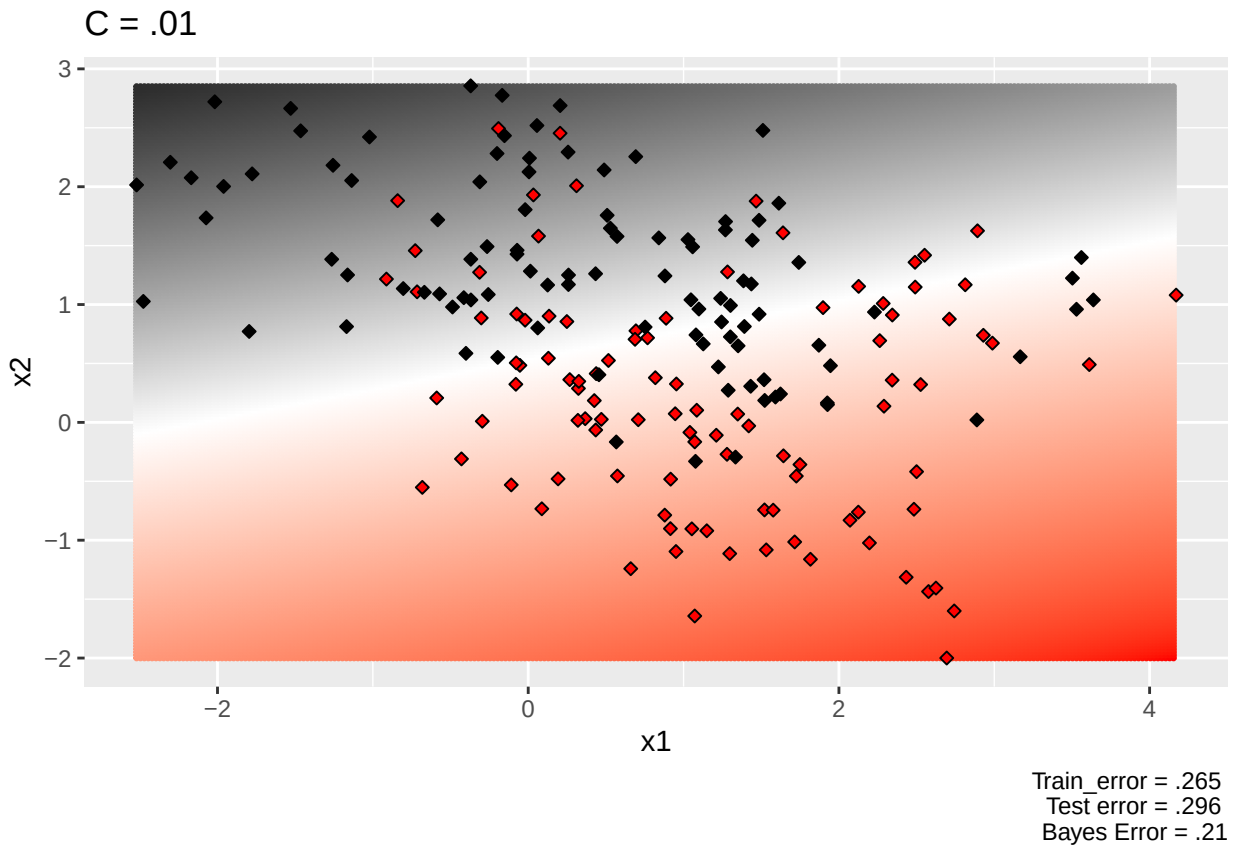
we will transform such probabilities with the logit function so that we can observe the uncertainty that our model assigns to each point in the feature space. This is for the model with cost $C = .01$

```
grid_1 <- grid %>% mutate(y = predict(results$model[[1]], grid, probability=TRUE),
                          y = logit(attr(y, 'probabilities')[,2]))

ggplot() +
    geom_point(data = grid_1, size=.5, aes(x1,x2,colour = y),show.legend = FALSE) +
    scale_colour_gradient2(low = "red", high ="black") +

    geom_point(data = mixture_train, shape=23, size=1.5, aes(x1,x2,fill = y),show.legend = FALSE)+
    scale_fill_manual(values=c("0"="red", "1"="black")) +

    labs(title = "C = .01", caption = "Train_error = .265
                                       Test error = .296
                                       Bayes Error = .21")
```

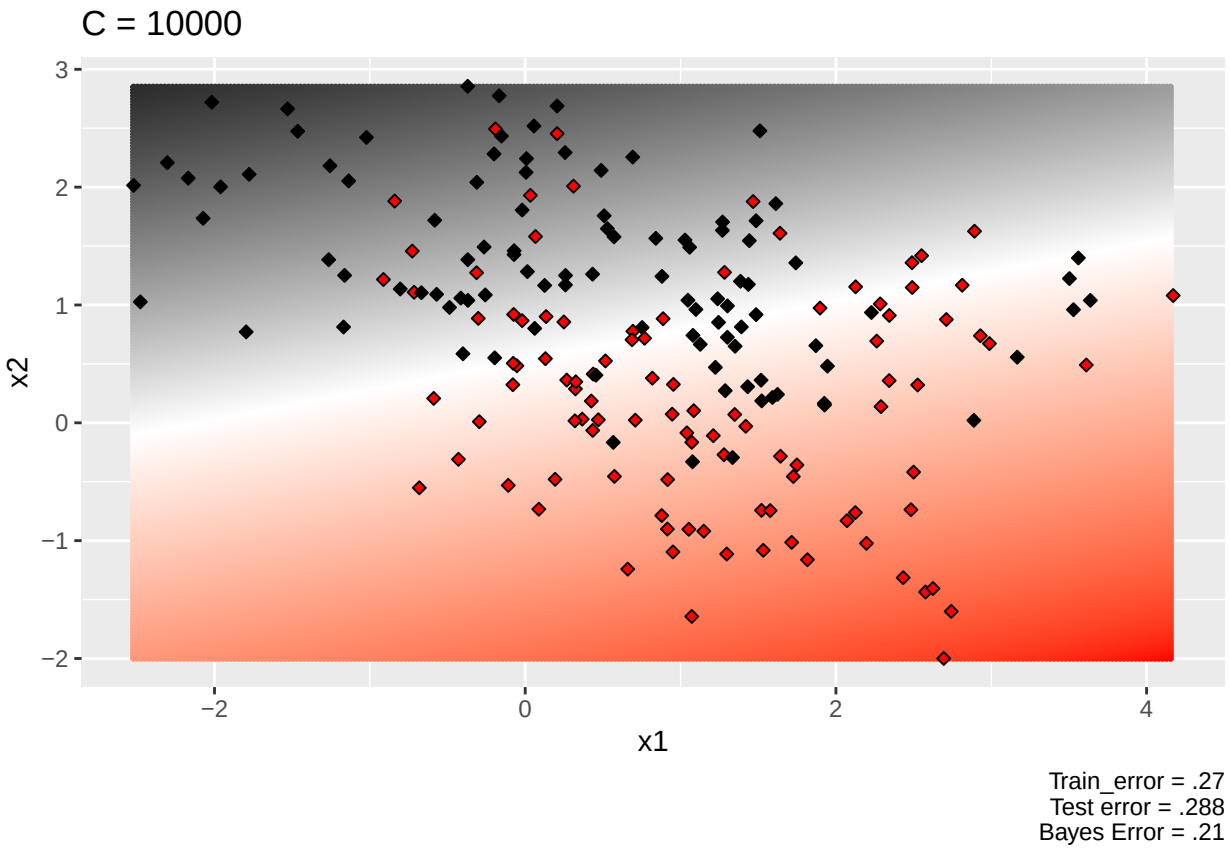


And now for thee model with cost $C = 10000$

```
grid_2 <- grid %>% mutate(y = predict(results$model[[2]], grid, probability=TRUE),
                          y = logit(attr(y, 'probabilities')[,2]))
ggplot() +
    geom_point(data = grid_1, size=.5, aes(x1,x2,colour = y),show.legend = FALSE) +
    scale_colour_gradient2(low = "red", high = "black") +

    geom_point(data = mixture_train, shape=23, size=1.5, aes(x1,x2,fill = y),show.legend = FALSE)+
    scale_fill_manual(values=c("0"="red", "1"="black")) +
```

```
    labs(title = "C = 10000", caption = "Train_error = .27
                                         Test error = .288
                                         Bayes Error = .21")
```



---

b) *At the time when you are working on this task, we have not yet discussed the "kernel trick" in the lecture to extend support vector machine to nonlinear decision boundaries. Nevertheless, we can still perform a "simpler" version of the nonlinear extension by considering an enlarged feature space with 5 coordinates* $x_1, x_2, x_1x_2, x_1^2, x_2$, *where* x_1, x_2 *are the original features of the data. Perform the same procedure of linear support vector classification (with* $C = 10000$ *and* $C = 0.01$*) as in part a) for the data in the enlarged feature space. Then plot the new nonlinear decision boundaries and their margins (i.e., those correspond to the dark dash lines in Fig. 12.2) together with the data points in the original* x_1 *vs* x_2 *plane. Do the new nonlinear decision boundaries look reasonable?*

The following functions recieves a data frame with x_1 and x_2 and outputs a dataframe containing the observations in the enlarged feature space with 5 coordinates x_1, x_2, x_1x_2, x_1^2, x_2^2,

```
convert_polynomial <- function(df){
  df %>% mutate(
            x1x2 = x1*x2,
            x1_squared = x1*x1,
            x2_squared = x2*x2
          )
}
```

We transform our tran and test datasets to the new 5 dimensional feature space

```
mixture_train <- convert_polynomial(mixture_train)
mixture_test <- convert_polynomial(mixture_test)
glimpse(mixture_train)
```

```
## Observations: 200
## Variables: 6
## $ x1         <dbl> 2.52609297, 0.36695447, 0.76821908, 0.69343568, -0.01983...
## $ x2         <dbl> 0.321050446, 0.031462099, 0.717486166, 0.777194034, 0.86...
## $ y          <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ x1x2       <dbl> 0.811003273, 0.011545158, 0.551186560, 0.538934074, -0.0...
## $ x1_squared <dbl> 6.3811456824, 0.1346555845, 0.5901605492, 0.4808530429, ...
## $ x2_squared <dbl> 1.030734e-01, 9.898637e-04, 5.147864e-01, 6.040306e-01, ...
```

And we build a dataframe `results` that, for each cost, trains the svm model and computes the train and test errors. Since the number of iterations was reached at $cost = 10000$ without a succesfull result, we'll use $cost = 9000$ that still works fine

```
results <- tibble(cost = c(.01,9000)) %>%
  mutate(
    model = map(cost, ~svm(y ~ ., mixture_train, type = 'C-classification',
                           kernel = 'linear', cost=.x, probability=TRUE, )),

    train_predicted = map(model, ~predict(.x, mixture_train)),
    train_actual = map(model, ~identity(mixture_train$y)),
    train_correct = map2_dbl(train_predicted, train_actual, ~sum(.x == .y)),
    train_error = (nrow(mixture_train)-train_correct)/nrow(mixture_train),

    test_predicted = map(model, ~predict(.x, mixture_test)),
    test_actual = map(model, ~identity(mixture_test$y)),
    test_error = map(model, ~get_test_error(.x))
  )
results$train_error
```

```
## [1] 0.28 0.27
```

```
results$test_error
```

```
## [[1]]
## [1] 0.293
##
## [[2]]
## [1] 0.279
```

As in a), we build the plot for the case $cost = .01$

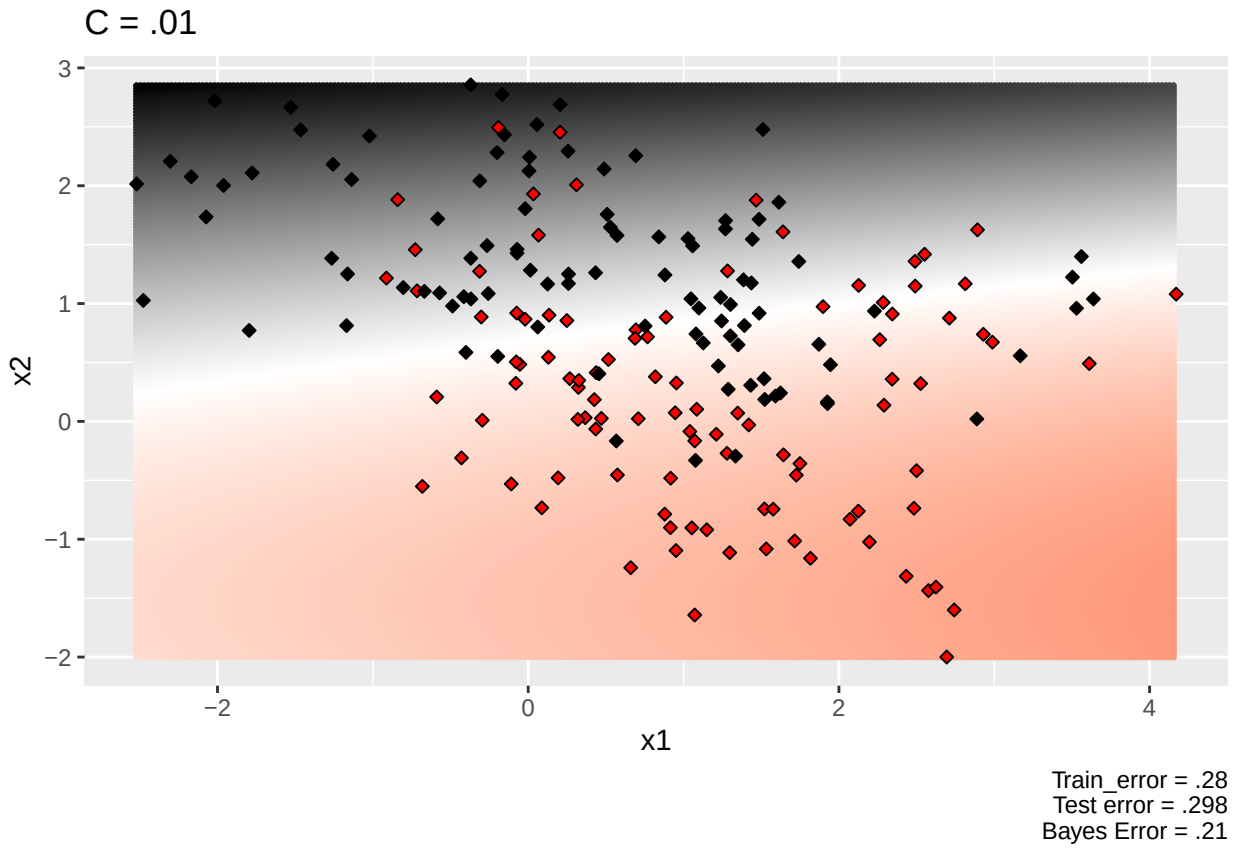
```
grid <- convert_polynomial(grid)
grid_1 <- grid %>% mutate(y = predict(results$model[[1]], grid, probability=TRUE),
                          y = logit(attr(y, 'probabilities')[,2]))

ggplot() +
    geom_point(data = grid_1, size=.5, aes(x1,x2,colour = y),show.legend = FALSE) +
    scale_colour_gradient2(low = "red", high = "black") +

    geom_point(data = mixture_train, shape=23, size=1.5, aes(x1,x2,fill = y),show.legend = FALSE)+
    scale_fill_manual(values=c("0"="red", "1"="black")) +

    labs(title = "C = .01", caption = "Train_error = .28
```

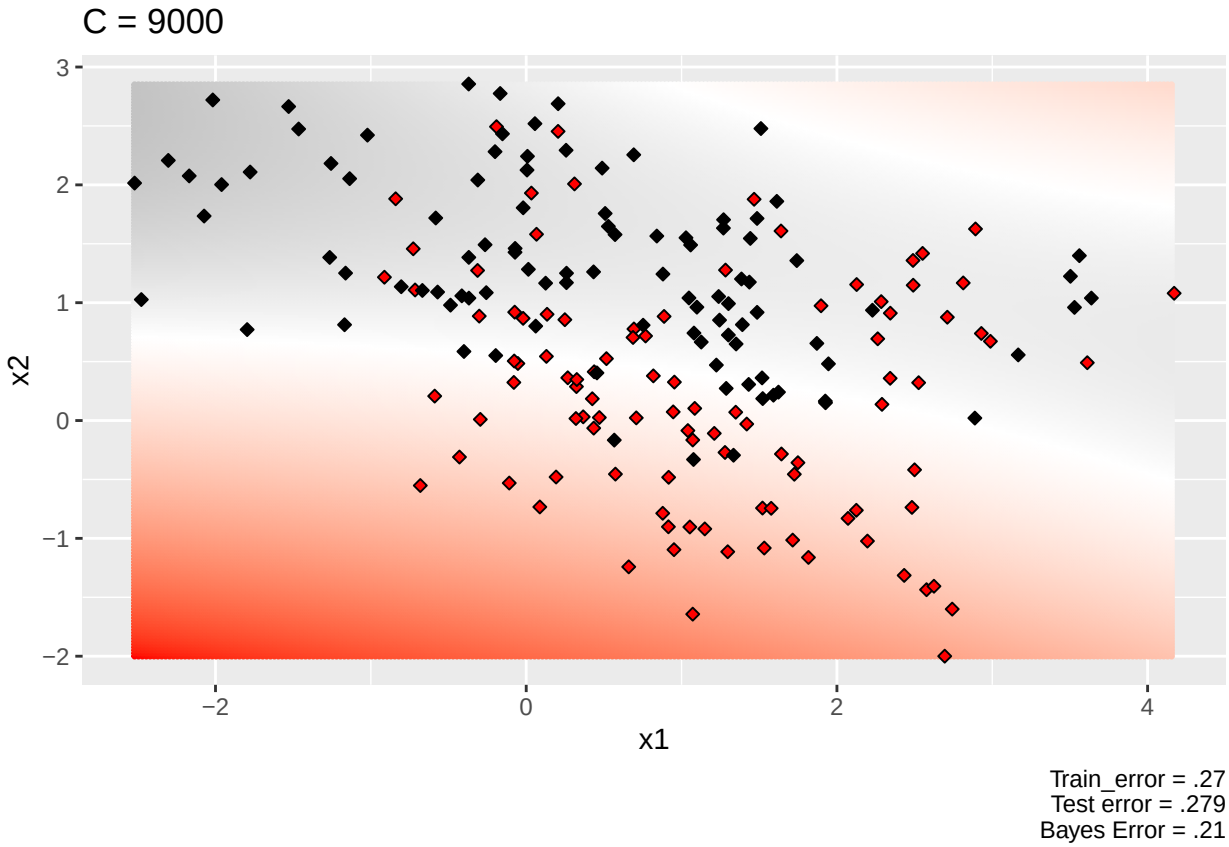
```
                                    Test error = .298
                                    Bayes Error = .21")
```



And now for $cost = 10000$

```
grid_2 <- grid %>% mutate(y = predict(results$model[[2]], grid, probability=TRUE),
                          y = logit(attr(y, 'probabilities')[,2]))
ggplot() +
    geom_point(data = grid_2, size=.5, aes(x1,x2,colour = y),show.legend = FALSE) +
    scale_colour_gradient2(low = "red", high = "black") +

    geom_point(data = mixture_train, shape=23, size=1.5, aes(x1,x2,fill = y),show.legend = FALSE)+
    scale_fill_manual(values=c("0"="red", "1"="black")) +
    labs(title = "C = 9000", caption = "Train_error = .27
                                       Test error = .279
                                       Bayes Error = .21")
```



The boundary of the first case (with cost $c = .01$) resembles a lot with the linear boundaries of the last task and the test error isn't any better. For the second case (with cost $c = 9000$), the boundaries are clearly non-linear and appear to create a good separation of the data achieving the best test error (.279) of all four clasifiers.